

Визуализация алгоритма определения числа Пи методом Монте-Карло с использованием Python

Болтовский Гавриил Александрович

Приамурский государственный университет им. Шолом-Алейхема

Студент

Аннотация

В данной статье рассмотрен алгоритм определения числа Пи методом Монте-Карло, его реализация на языке программирования Python, а также визуализация данного алгоритма. Разработана программа на Python с использованием модуля PyGame для демонстрации работы алгоритма приближённого вычисления числа Пи методом Монте-Карло, а также предложен способ оптимизации программной части на вычислениях при больших количествах точек на плоскости.

Ключевые слова: Python, математические алгоритмы, система Лоренца, метод Монте-Карло, число Пи

Visualization of the Monte Carlo method for calculating Pi using Python

Boltovsky Gavriil Alexandrovich

Sholom-Aleichem Priamursky State University

Student

Abstract

This article discusses the algorithm for determining the number Pi using the Monte Carlo method, its implementation in the Python programming language, and the visualization of this algorithm. A Python program was developed using the PyGame module to demonstrate the operation of the algorithm for approximate calculation of Pi using the Monte Carlo method, and a method was proposed for optimizing the software part for calculations with large numbers of points on a plane.

Keywords: Python, mathematical algorithms, Lorenz system, Monte Carlo method, Pi

1. Введение

1.1 Актуальность

Метод Монте-Карло – один из самых влиятельных и широко применяемых вычислительных подходов в науке, инженерии, финансах и технологиях. Его актуальность в 2020-х годах не только не снижается, но и растёт благодаря развитию вычислительной техники, появлению больших данных и сложных стохастических моделей.

Многие реальные системы слишком сложны для точного математического описания, а метод Монте-Карло позволяет оценивать различные величины численно, даже если их нельзя выразить формулой.

Правильная программная реализация метода Монте-Карло позволяет использовать её в различных областях: от математической статистики до машинного обучения и робототехники.

1.2 Обзор исследований

В.Н. Акмаева предложила обобщение алгоритма поиска гарантирующего решения задачи квантильной оптимизации [1]. В основе исследования закладывается метод Монте-Карло для генерации и разметки случайной выборки и метод опорных векторов.

Е.Ю. Елизарова и М.А. Волкова в своей работе рассмотрели модель интерфейса приложения «Численные методы для школьников», где более подробно описана страница по визуализации метода Монте-Карло, являющегося частью электронного курса [2].

Р.А. Перепелица и И.А. Тарасова проанализировали в своей статье наиболее популярные методы оценки и анализа проектных рисков на предприятии, а именно деревья принятия решений и метод Монте-Карло [3]. Были выделены преимущества и недостатки каждого метода.

А.П. Кашникова и М.Б. Беляева рассмотрели понятие метода Монте-Карло и способы его применения [4]. В статье также представлен разбор примера в среде Matlab.

1.3 Цель исследования

Цель исследования – разобрать и написать алгоритм приближённого вычисления числа Π методом Монте-Карло, разработать графическую программу для визуализации рассмотренного метода.

2 Материалы и методы

Для решения поставленных задач выбран язык программирования Python. Для визуализации алгоритма выбран сторонний модуль PyGame.

3 Результаты и обсуждения

Для достижения наилучшего результата работы алгоритма стоит разобрать метод Монте-Карло.

Метод Монте-Карло – это статистический подход к решению задач с использованием случайных выборок. Он особенно полезен, когда аналитическое решение сложно или невозможно. Один из классических примеров применения метода Монте-Карло – приближённое вычисление числа Π .

Идею метода можно сформулировать следующим образом:

Рассмотрим квадрат со стороной 2, в который вписан круг радиуса 1 (с центром в начале координат).

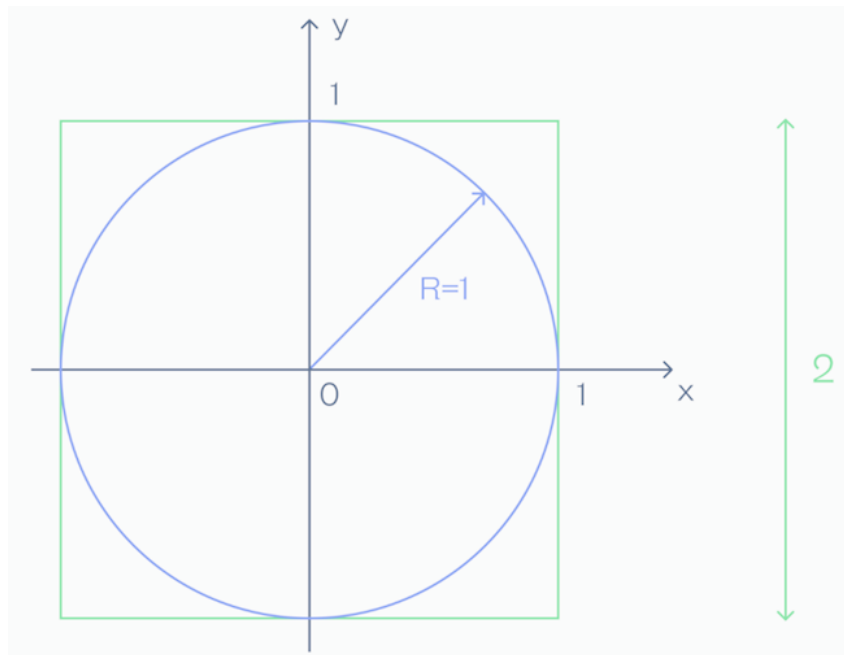


Рисунок 1. Визуализация задачи приближённого вычисления числа Пи

Площадь квадрата будет равна 4, а площадь круга равна Пи соответственно. Если случайно «ставить точки» в этот квадрат, то вероятность того, что точка попадёт внутрь круга, равна отношению площадей:

$$P = \frac{S_{\text{круга}}}{S_{\text{квадрата}}} = \frac{\pi}{4}$$

Следовательно, если сгенерировать большое количество случайных точек (x, y) в квадрате $[-1, 1] \times [-1, 1]$, и посчитать, сколько из них попало внутрь круга (т.е. удовлетворяют условию $x^2 + y^2 \leq 1$), то можно оценить π по формуле:

$$\pi \approx 4 * \frac{N_{\text{внутри}}}{N_{\text{всего}}}$$

Где $N_{\text{внутри}}$ это количество точек, попавших внутрь круга.

Решение данной задачи можно легко перенести на любой существующий язык программирования за счёт простоты своей реализации. Для подтверждения данного утверждения был написан алгоритм приближённого вычисления числа Пи методом Монте-Карло:

```
def estimate_pi(n):
    inside = 0
    for _ in range(n):
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)
        if x**2 + y**2 <= 1:
```

```
        inside += 1
    return 4 * inside / n
```

При увеличении значения входного параметра n можно достигнуть более точного значения числа Π , но при этом возрастает и время выполнения программы (см. рис. 2).

```
n = 1000 => Pi = 3.176
n = 10000 => Pi = 3.1472
n = 100000 => Pi = 3.13676
n = 1000000 => Pi = 3.142624
n = 10000000 => Pi = 3.1413912
n = 100000000 => Pi = 3.14142212
```

Рисунок 2. Результаты работы алгоритма приближённого вычисления числа Π на различных входных данных

Процесс приближённого числа можно визуализировать, если отмечать точки на координатной плоскости определёнными цветами: зелёным, если точка попала внутри круга, и красным в обратном случае. Для визуализации будем использовать сторонний модуль PyGame.

Было решено разработать программу с графическим интерфейсом с предоставлением возможности интерактивного взаимодействия посредством изменения количества генерируемых точек. На начальном этапе были определены глобальные переменные, задающие размеры окна приложения и квадрата на плоскости, радиус круга, а также палитру цветов в приложении:

```
WIDTH, HEIGHT = 1200, 1000
INFO_WIDTH = 250

FIELD_WIDTH = WIDTH - INFO_WIDTH

SQUARE_SIZE = min(FIELD_WIDTH, HEIGHT)
SQUARE_LEFT = INFO_WIDTH
SQUARE_TOP = 0

RADIUS = SQUARE_SIZE // 2
CENTER = (SQUARE_LEFT + SQUARE_SIZE // 2, SQUARE_TOP + SQUARE_SIZE // 2)

WHITE = (255, 255, 255)
GRAY = (100, 100, 100)
BLACK = (0, 0, 0)
BLUE = (0, 0, 255)
GREEN = (0, 200, 0)
RED = (200, 0, 0)
YELLOW = (255, 255, 0)
DARK_GRAY = (40, 40, 40)
```

Само приложение включает в себя 2 основные области на экране: панель управления слева, в которой отображается приближённое значение числа Пи, а также поле для ввода количества точек и кнопка начала генерации (см. рис. 3).

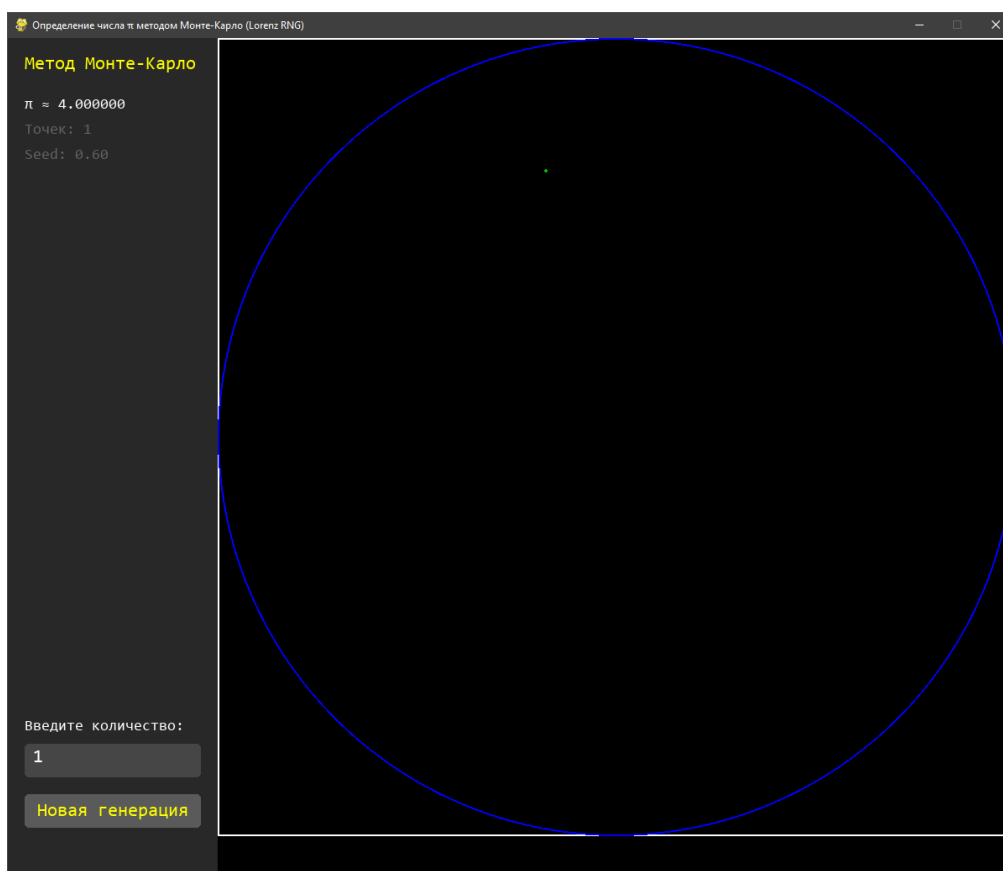


Рисунок 3. Интерфейс приложения приближённого вычисления числа Пи

Для генерации случайных чисел можно использовать как встроенный генератор псевдослучайных чисел, так и реализованный вручную. В рамках данной статьи был использован ГПСЧ на основе системы Лоренца.

Система Лоренца – это система из трёх нелинейных дифференциальных уравнений, впервые предложенная Эдвардом Лоренцем в 1963 году для упрощённого математического описания атмосферной конвекции. Эта система стала одной из наиболее известных в теории хаоса и демонстрирует сложное, хаотическое поведение при определённых значениях параметров.

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = x(\rho - z) - y \\ \dot{z} = xy - \beta z \end{cases}$$

Рисунок 4. Математическое описание системы Лоренца

В уравнениях системы Лоренца задействованы следующие переменные:

1. x, y, z – переменные, зависящие от времени t .
2. σ – число Прандтля, характеризующее соотношение вязкости и теплопроводности.
3. ρ – число Рэлея, описывающее меру разности температур между верхним и нижним слоями жидкости или газа.
4. β – геометрический параметр, связанный с размерами области.

На основе ранее написанного алгоритма приближённого вычисления числа Π был написан новый, включающий в себя использование разработанного ГПСЧ на основе системы Лоренца. Функция на вход получает количество точек и специальное значение для генерации случайных чисел `seed`, затем в цикле ставит точки в случайном месте и проверяет расположение точки относительно круга. Функция возвращает список точек и найденное приближённое значение числа Π :

```
def generate_points(n, seed=0.5):
    rng = LorenzRNG(seed=seed)
    points = []
    circle_points = 0
    for _ in range(n):
        x = SQUARE_LEFT + rng.next() * SQUARE_SIZE
        y = SQUARE_TOP + rng.next() * SQUARE_SIZE

        dist2 = (x - CENTER[0]) ** 2 + (y - CENTER[1]) ** 2
        inside = dist2 <= RADIUS ** 2
        if inside:
            circle_points += 1
            points.append((x, y, inside))
    pi_estimate = 4 * circle_points / n
    return points, pi_estimate
```

В программе реализована событийная модель взаимодействия пользователя с интерфейсом. В зависимости от нажатий пользователя программа может по-разному реагировать, в основном обрабатываются события запуска генерации и изменения значения в поле для ввода количества точек:

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False

    if event.type == pygame.MOUSEBUTTONDOWN:
        if input_box.collidepoint(event.pos):
            input_active = True
        else:
            input_active = False

    if button_gen.collidepoint(event.pos):
        try:
```

```
TOTAL_POINTS = max(0, int(user_input))
except ValueError:
    TOTAL_POINTS = 100_000
seed_value += 0.1
points, pi_estimate = generate_points(TOTAL_POINTS,
seed=seed_value)
shown_points = 0
visualizing = True

if event.type == pygame.KEYDOWN and input_active:
    if event.key == pygame.K_RETURN:
        input_active = False
    elif event.key == pygame.K_BACKSPACE:
        user_input = user_input[:-1]
    elif event.unicode.isdigit():
        user_input += event.unicode
```

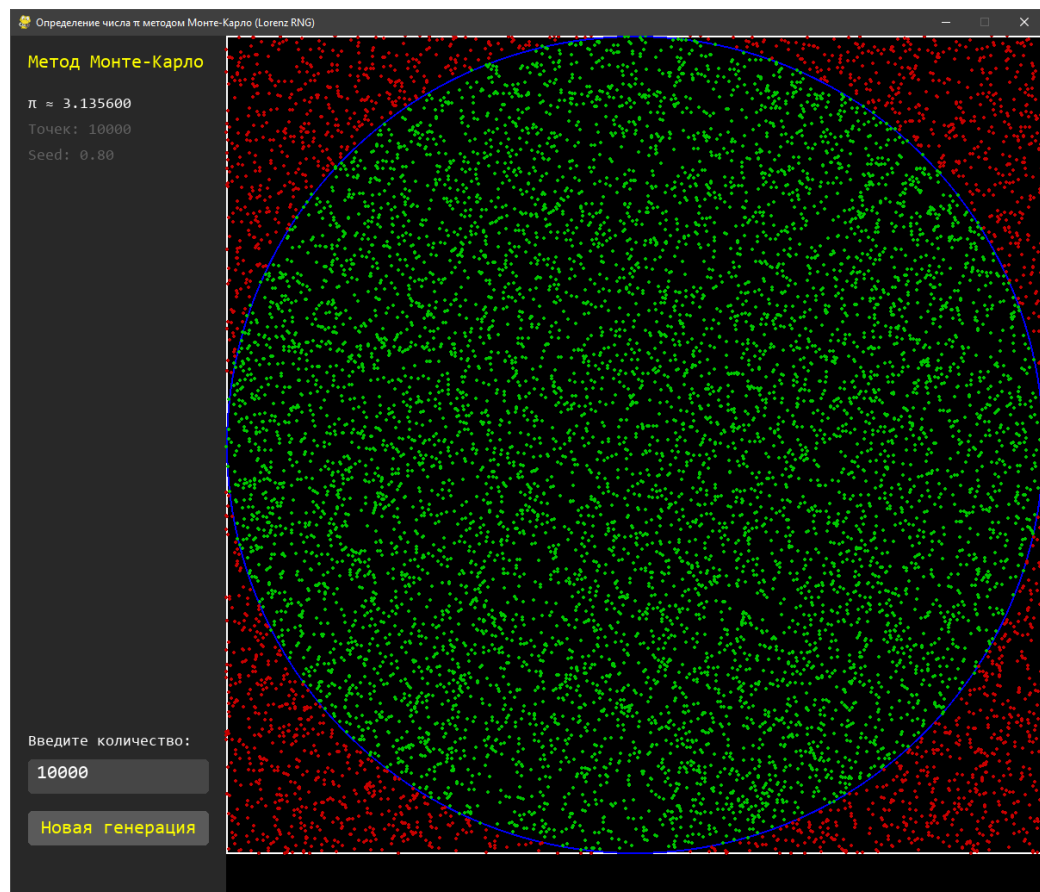


Рисунок 5. Пример работы программы приближённого вычисления числа π методом Монте-Карло

Также можно рассмотреть возможность параллельной обработки нескольких точек одновременно для повышения скорости работы программы на больших количествах точек. Основная проблема заключается в том, что даже при $n = 1.000.000$ программа начинает долго проводить расчёты, что в реальных условиях недопустимо.

В случае описанного алгоритма генерация точек – идеальный кандидат для распараллеливания, так как каждая точка независима. Ниже представлена мультипроцессорная реализация генерации точек на плоскости, предполагающая деление количество точек на несколько частей. Этот подход позволяет ускорить генерацию почти пропорционально количеству ядер процессора.

В новой реализации

1. Общее количество точек n разбивается на несколько частей.
2. Каждая часть обрабатывается в отдельном процессе с помощью модуля multiprocessing.
3. Количество процессов автоматически определяется по числу ядер CPU (но ограничено сверху значением 8 для стабильности).
4. Каждому процессу передаётся уникальный начальный параметр (seed), чтобы избежать дублирования последовательностей.
5. Все процессы одновременно генерируют свои подмножества точек и подсчитывают, сколько из них попали внутрь круга.
6. После завершения работы все результаты собираются в главном процессе, суммируются, и вычисляется итоговая оценка Π .

```
def generate_points_chunk(start_idx, end_idx, seed, params):
    rng = LorenzRNG(seed=seed)
    points = []
    inside_count = 0

    square_left = params["square_left"]
    square_top = params["square_top"]
    square_size = params["square_size"]
    center_x, center_y = params["center"]
    radius = params["radius"]
    radius_sq = radius * radius

    for i in range(end_idx - start_idx):
        x = square_left + rng.next() * square_size
        y = square_top + rng.next() * square_size

        dx = x - center_x
        dy = y - center_y
        dist_sq = dx*dx + dy*dy
        inside = dist_sq <= radius_sq

        if inside:
            inside_count += 1

        points.append((x, y, inside))

    return points, inside_count

def generate_points_parallel(n, seed=0.5, num_processes=None):
    if n <= 0:
```

```
        return [], 0.0

    if num_processes is None:
        num_processes = min(mp.cpu_count(), 8)
    params = {
        "square_left": SQUARE_LEFT,
        "square_top": SQUARE_TOP,
        "square_size": SQUARE_SIZE,
        "center": CENTER,
        "radius": RADIUS
    }

    chunk_size = max(1, n // num_processes)
    chunks = []
    for i in range(num_processes):
        start_idx = i * chunk_size
        end_idx = min((i + 1) * chunk_size, n) if i < num_processes
    - 1 else n
        if start_idx < end_idx:
            chunks.append((start_idx, end_idx, seed + i * 0.01))

    with mp.Pool(processes=num_processes) as pool:
        results = pool.starmap(
            partial(generate_points_chunk, params=params),
            chunks
        )

    all_points = []
    total_inside = 0

    for points_chunk, inside_count in results:
        all_points.extend(points_chunk)
        total_inside += inside_count

    pi_estimate = 4 * total_inside / n if n > 0 else 0.0
    return all_points, pi_estimate
```

4 Выводы

Таким образом была разработана программа для визуализации алгоритма нахождения приближённого числа Π методом Монте-Карло, которую можно использовать в образовательной деятельности для демонстрации описанного алгоритма.

Библиографический список

1. Акмаева В.Н. Метод Монте-Карло и метод опорных векторов для решения задач квантильной оптимизации // Материалы XXIV Международной конференции по вычислительной механике и современным прикладным программным системам (ВМСППС'2025), Алушта, 07–13 сентября 2025 года. Москва: Федеральное государственное бюджетное образовательное

- учреждение высшего профессионального образования "Московский авиационный институт (национальный исследовательский университет)". 2025. С. 273-274.
2. Елизарова Е.Ю., Волкова М.А. Моделирование интерфейса приложения для визуализации метода Монте-Карло в курсе "Численные методы для школьников" // Муниципальное образование: инновации и эксперимент. 2021. № 2(77). С. 4-11.
 3. Перепелица Р.А., Тарасова И.А. Разработка метода оценки проектных рисков на базе деревьев принятия решений и метода Монте-Карло // Современные информационные технологии в образовании и научных исследованиях (СИТОНИ-2021): Материалы VII Международной научно-технической конференции, Донецк, 23 ноября 2021 года / Под общей редакцией В.Н. Павлыша. Донецк: Донецкий национальный технический университет. 2021. С. 248-252.
 4. Кашникова А.П., Беляева М.Б. Метод Монте-Карло в задачах моделирования процессов и систем // Modern Science. 2021. № 1-2. С. 358-362.